An Efficient Resource Monitoring Service for Fog Computing Environments

Sudheer Kumar Battula, Saurabh Garg, *Member, IEEE,* James Montgomery, *Member, IEEE,* and Byeong Kang, *Member, IEEE*

Abstract—With the increasing number of Internet of Things (IoT) devices, the volume and varieties of data being generated by these devices are increasing rapidly. Cloud computing cannot process this data due to its limitations such as latency and scalability. In order to process this data in less time, Fog computing has evolved as an extension to Cloud computing. In a Fog computing environment, a resource monitoring service plays a vital role in providing advanced services such as scheduling, scaling, and migration. Most of the works in Fog computing have assumed that a resource monitoring service is already available. However, there are not any studies, which investigate resource monitoring service in detail for Fog computing environments. Conventional methods proposed for other distributed systems such as Cloud and Grid may not be suitable due to the unique features of a Fog environment such as the limited capacity and heterogeneity of Fog devices. To improve the overall performance of Fog computing and optimise resource usage, effective resource monitoring techniques are required. Hence, we propose a Support and Confidence (SCB) based technique, which optimises the resource usage in the resource monitoring service. The performance of our proposed system is evaluated by examining a real-time traffic use case in a Fog emulator and the results are compared with traditional distributed computing techniques. The experimental results obtained from the Fog Emulator show that the proposed technique consumes fewer resources compared to conventional resource monitoring approaches.

Index Terms—Fog Computing, Resource Monitoring, Internet of Things.

1 Introduction

The number of IoT devices is increasing gradually and these devices are used in different vertical marketplaces for the benefit of the organisations with the help of IoT applications. These applications include energy distribution (smart energy grids) [1], smart cities (smart homes, buildings, transportations, highway systems, traffic management systems, and parking systems) [2], shipping (smart cargo tracking systems) [3], agriculture, and health [4]. According to Cisco, in 2020 more than 50 billion IoT devices will exist and be connected to the information space [5]. The amount of data generated from these devices is huge. Cloud computing has been considered as the main enabler to process big data generated by IoT devices. However, due to the limitations of Cloud computing in terms of latency and bandwidth, Cloud computing based systems are unable to support many time-sensitive IoT applications. Hence, Fog computing has evolved to address these limitations. Fog computing is a distributed computing paradigm where data is stored and processed closely at the end devices [6]. All of these numerous IoT devices inherently form a massively distributed environment. Hence, IoT infrastructure is more complex and larger in scale. In order to manage this complex infrastructure, efficient Fog services are required to improve the performance of the system. Among all services, resource monitoring plays a vital role in the performance of any system as it is a required service for all other advanced services such as scheduling, scaling and migration to work, as shown in Figure 1.

Monitoring is a process of tracking and collecting information about computing resources and reporting the status of the computing devices to the subscriber or controller of the system. The resources of the system are CPU, RAM, bandwidth, and battery. In general, monitoring services include security monitoring, application monitoring, and resource monitoring.

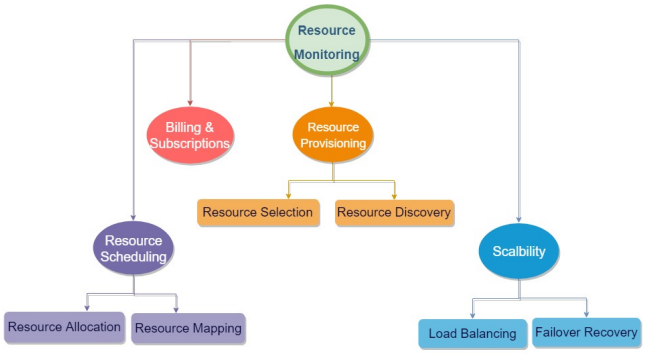


Fig. 1: Resource monitoring dependency services.

Efficient resource monitoring can deliver the resource information in a precise manner which ensures the robustness of the system (by detecting the over usage and under usage of the system and taking necessary actions), accurate billing (by maintaining the information of resource allocation and consumption of applications), guaranteed Quality of Service (QoS) and Service Level Agreement (SLA) (by ensuring the availability of resources through disaster recovery and backups), fault management (by detecting and handling the failures of resources) and cost-effectiveness to the users. Delicato et al. [7] and Perera et al. [8], identified the essential requirements, challenges, and peculiarities involved in monitoring the resources of Fog computing environment.

In the recent literature on Fog, a few researchers have

developed deployment services, scheduling, and other advanced services for the Fog computing paradigm by assuming the resource monitoring services are already available. Most of the available research works on Fog computing do not analyse the performance of resource monitoring service. In [9] and [10] researchers built Fog computing platforms and emulators with deployment and scheduling services by assuming that the resource availability characteristics are inbuilt, which is not true. Hence, efficient resource monitoring by optimizing the resource utilisation in resource monitoring is a still an open issue. The challenges in providing a resource monitoring service in Fog environment include working with limited resources such as bandwidth, battery power, computational power, and storage.

Moreover, most of the previous research works on resource monitoring were carried out in Grid, Cluster and Cloud computing. Currently, many Cloud providers in the market use their own proprietary monitoring tools for the Clouds to efficiently provide services to the users. Cloud Watch [11] by Amazon, Azure Watch and Cloud Monix [12] by Microsoft Azure and IBM Tivoli [13] by IBM cloud are some examples of resource monitoring tools. Although in the literature, many researchers proposed various resource monitoring techniques for Cloud computing systems, these systems and techniques are difficult to use in Fog environment due to the limited capacity-Fog devices, brisk interactions and the diverse characteristics of devices, platforms, protocols and data formats.

To address this gap, this paper investigates the performance of traditional resource monitoring approaches in a Fog environment and provides an efficient resource monitoring service which is currently non-existent to the best of our knowledge.

The contributions of this study are as follows:

1) A comparative analysis of resource monitoring service between traditional approaches in terms of utilisation of CPU, battery, and bandwidth of Fog devices and Fog leaders within the proposed Fog emulator.
2) An Support and Confidence (SCB) Based approach for efficient resource monitoring service with minimum consumption of resources such as bandwidth, battery and processing power of Fog devices.

The paper is organized into four sections. Section 2 reviews related works of resource monitoring techniques in distributed systems. Section 3 discusses the system model in detail. We discuss the proposed resource monitoring technique in Section 4. Section 5 investigates the existing resource monitoring models. The final section deals with the results and conclusions.

2 Related Work

Resource monitoring holds a very crucial role in the performance of the system. Research in resource monitoring has a long history in distributed systems such as Grid, Cloud and other distributed systems, but there have been a few studies in Fog computing. Naha et al. [14] suggested that the resource monitoring should able to detect the future needs of the applications to avoid the SLA violation. Liyanage et al. [15] has given a foresight study on Fog/Edge infrastructures with requirements and specifications of Fog/Edge monitoring service. The study presented a classification of monitoring architectures, services, and tools of existing Cloud and distributed systems and suggested that our community should propose effective monitoring service in Fog/Edge computing.

But much of the literature on Fog computing paid less attention to resource monitoring service and mainly focused in providing advanced services such as deployment services, scheduling, and other advanced services by assuming that the basic services of resource monitoring are provided by default. Vasconcelos et al. [6] did not consider the resource monitoring feature in their proposed system. In another study, Tsai et al. [16] developed a deployment service by using kuberentes [1] as the resource monitoring service. Gupta et al. [17] developed a Fog simulator with different policies of resource management which had a component for monitoring, but the study did not specify how the monitoring was done in the simulator. Other researchers assume that the existing resource monitoring techniques of another field can be used in Fog computing. Souza et al. [18] formulated the service assignment as an optimisation problem in the Fog to Cloud scenario and met the QoS requirement but they assumed that resource monitoring was provided at the cloud. Aazam and Huh [19] proposed a service-oriented resource management system and a cost model based on the reservation for the customers by assuming the resource monitoring service is available. Therefore, many researchers are not considered resource-monitoring studies in this area.

The following subsections discuss the resource monitoring techniques and tools in the other related fields such as grid, cloud and sensor networks.

2.1 Grid monitoring approaches and tools

The taxonomy for Grid monitoring approaches has proposed and categorised in four levels starting from 0 to 3 based on their characteristics [20]. The four levels are described below:

1) **Self-contained systems:** in self-contained systems, events are sent directly from sensors to consumers with no intermediates.
2) **Producer only systems:** in these systems, sensors are either loosely or tightly connected to publishers (hosts) and the events are passed through intermediate components to consumers.
3) **Producer and republisher systems:** these systems have an intermediate component which is responsible for collecting the data from the producers and sending them to the consumers.
4) **Hierarchy of republishers:** these systems have one or more intermediate components acting as both producers and consumers, following a strict hierarchy to send the events from sensors to the consumer.

Sundaresan et al. [21] proposed a producer and republisher based architecture for monitoring in which they used a pull model and coherence protocol to maintain the

1. https://kubernetes.io/

resource information up to date. Chung et al. [22] developed three mechanisms to monitor the resources based on the push-based model for grid systems. Czajkowski et al. [23] proposed Grid Resource Information Monitoring (GRIM) to continuously monitor the resources of grid and client queries for the resources in order to provide resources in minimum time. Grid Global Forum (GGF) was proposed as a grid monitoring service architecture [24] based on the producer, consumer and directory model for effective resource monitoring. Foster et al. [25] compared cloud and other distributed technologies such as grid, cluster and utility computing in different aspects and concluded that many features are common in both cloud and grid computing. With this motivation, many researchers [26], [27] converted grid monitoring tools to cloud monitoring tools.

However, the resource monitoring tools in the grid are not designed for limited resource devices and thus may consume excessive resources and create a burden for Fog devices.

2.2 Cloud computing resource monitoring

Many Grid resource monitoring tools are extended for cloud resource monitoring. Traditionally monitoring systems are centralised models due to the limitations such as a single point of failure and bottlenecks through which it causes degradation of the performance in the system. To overcome this problem, Xu et al. [28] proposed a distributive collaborative monitoring model and claimed that this model could provide rapid notifications and recovery under tainted conditions. In [29] researchers proposed agent-based resource monitoring frameworks and techniques for Infrastructure as a Service (IaaS) cloud. The agent would send the resource information of available computing resources to the scheduler and then with available information scheduler would take appropriate decision. Aneka [30] cloud framework and Nimsoft [31] have their own middleware component for monitoring both applications and resources which supports public, private and hybrid clouds.

There are many open source monitoring tools to monitor the resources in cloud environments. DARGOS [32] is a distributed monitoring tool, built based on a hybrid model to detect and balance the overhead of the virtual and physical resources easily in multiple zones. Private Cloud MONitoring System (PCMONS) [33] has seven different components, each responsible for monitoring, configuring generating and visualising of the system. The other resource monitoring tool used in Cloud stack is Zensos extension (Zenpack) [34] where it provided the alerts and events about the core networking and system. Open Nebula [35] used an information module manager for resource monitoring which provides the status of the physical devices. There are other tools such as cloudharmony [36] and cloudstone [37] which provides the performance evaluation of different clouds through benchmark programs. Although in the literature, many researchers have proposed various resource monitoring tools and techniques for cloud computing systems, these are difficult to use in Fog environment because of the following reasons:

1) In Cloud computing, the tools are built for computing environment with large and unlimited resources, but the Fog computing has limited resources.
2) In Cloud computing, assigning tasks (scheduling) and other advanced services are based on the billing and priority subscriptions but the time-sensitive applications in Fog computing demand a fair monitoring policy.
3) Resource monitoring always has some overhead on device-level exploration as the resources are limited. Mainly, the migration service demands huge resources, but the resources are limited in Fog computing.

2.3 Sensor mobile networks resource monitoring

There are many mobile sensor network tools and models that exist in the literature such as Eon [38] and Pixie [39]. But these tools do not provide a global or holistic view of resources. Kang et al. [40] claimed that the proposed orchestrator could provide a holistic view and find appropriate application resource usage and flexibly utilise resources in dynamic conditions by using plan selection, generation, and execution. However, the authors used a fixed number of sensors for evaluating the dynamic conditions and performed resource monitoring continuously to update the resource status of mobile devices and sensors. Li et al. [41] proposed a middleware called NonStop which clusters the nodes that had similar patterns and predicted the node to replicate the multimedia data and provided continuous data streaming to the devices. Hu and Johnson [42] proposed a technique based on the network layer information which extracted network flow information and analysed locally to find the behaviour of the system. They were unable to find the resource usage information from this technique.

2.4 Summary

Many researchers have contributed to resources monitoring in the distributed systems and presented the solutions for the cloud, grid and cluster systems which are best suited for their respective systems. However, due to differences in the infrastructure and characteristics of Fog computing such massive distribution of resources, reconsideration of resource monitoring techniques is required.

Previous studies in Fog computing have not addressed the answers to the following questions to the full degree.

1) Can we apply the resource monitoring techniques of another field into Fog computing? If yes, which technique should we follow for efficient resource monitoring?
2) How frequently do we need to monitor resources and when do we need to monitor resources?

Some Fog devices are non-stationary objects and the resources such as battery power, CPU, storage, and bandwidth are very limited. It is not a good idea to keep track of resources continuously as each Fog device status request consumes system resources. The other way is to update the resources upon the time of scheduling and rescheduling, which delays the execution of services. Fog computing mainly deals with the time-sensitive applications. Therefore, finding the best solution to keep track of resources is really

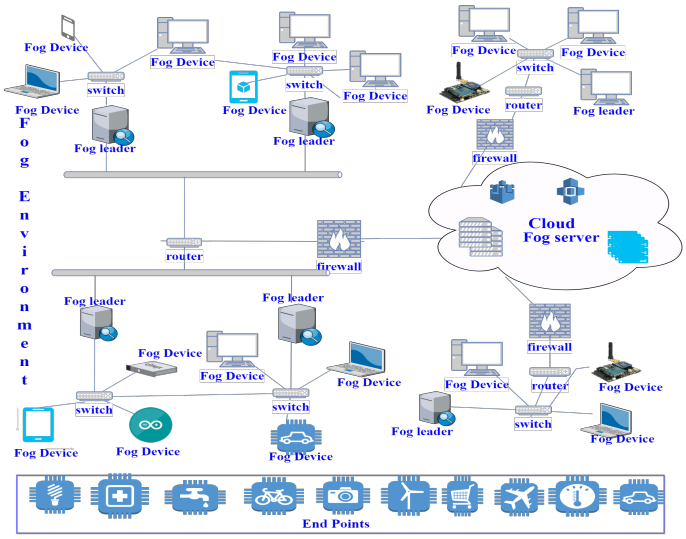


Fig. 2: Fog Environment

an important challenge in limited resource Fog environments. In this regard, we address the following research questions in this paper:

1) What will be the impact and overhead of the resources by traditional passive and active monitoring techniques in limited resource Fog environments?
2) How can we monitor the resources with minimum utilisation of resources in Fog environments?

3 Fog Model

This section presents a detailed description of the Fog model. As per the Open Fog consortium [43], we considered Fog environment which comprises of one Fog Server and one or more Fog Leaders for each Fog colony with multiple Fog devices and sensors under each leader of that particular Fog colony as shown in Figure 2.

The following assumptions are considered in Fog model.

1) A Fog device will always be ready to participate if it is available until it is unregistered.
2) Fog devices will have in-built security mechanisms to protect the collected data from unauthorised access and run on battery power.

3.1 Fog server

The Fog server is located in the Cloud and is responsible for creating and managing the environment, accepting service request from the users and forwarding the requests to the Fog Leader as well as managing the faults of the Fog leader.

3.2 Fog Leader

The Fog leader accepts a service activation request from the Fog server and sends the service activation request to the Fog devices and sensors. The number of Fog devices will be selected based on the number of sensors available in that particular colony. The Fog leader selects the devices and sends requests to these devices with service name and the number of containers it should run. The Fog leader collects and aggregates the output of the service request from the device and sends the final result to the requested user.

3.3 Fog Device

The Fog device accepts the service activation requests, from the Fog leader and creates the number of containers[2] requested by the Fog leader. Each container collects the data from the sensors and runs the service requested task on the collected data. After successful completion, it sends results back to the leader. In Fog environment, different types of Fog Devices exist such as Meshlium or Raspberry Pi, Mobile devices, Laptop, and server. These devices are categorised as tiny, small, medium and large based on the configuration of the systems as specified in Fog Computing Surveys [8].

3.4 Sensors

Sensors accept the requests from the Fog leader and send the data to the assigned Fog devices. The sensor sends the number of messages to the fog devices and calculates delays based on the protocol and network type of the sensor.

Fog user requests the Fog server to activate the service by specifying the service name and location. The Fog Server accepts the requests from Fog users and sends them to the Fog Leader of that particular location. The Fog leader will search for appropriate sensors for that service in that location and dynamically assigns Fog devices based on the number of sensors, from the live node list which has minimum requirements to execute the service. Retrieving the live node list will be explained with different approaches in the next section. This paper aims to investigate the resource monitoring, so we implemented a basic scheduling service.

For each service task, processing and monitoring resources such as CPU, network, RAM, battery power will be consumed. The delay and resource consumption values are presented by running the OS profiling benchmark programs such as *sysbench* [3] to find number of events that can be processed in a second, and the *powertop* [4] to find the power usage of the application, and the *top* process in Linux to collect the resource information in the actual systems with and without containers. Battery and network consumption values are taken from [44] and the summary of all values are shown in Table 1 and 3.

4 Proposed SCB Resource Monitoring Algorithm

In the proposed model, the resource monitoring agent will run in three levels such as Fog server, Fog leader and Fog device. Fog Server agent application predicts the liveliness of Fog devices and resource information from the time series historical data of Fog devices. The predictions are based on the probability of liveliness of a particular Fog device in a specific location at that particular time. This technique is known as "Support and Confidence" (SCB). Fog leader agent application will assign services to predicted devices and collects the results and resource information of Fog devices. Fog device agent application will collect and sends the resource information whenever it receives resource information request from the Fog server. The proposed SCB algorithm optimises the resource utilisation of Fog devices

2. https://www.docker.com/resources/what-container
3. http://manpages.ubuntu.com/manpages/bionic/en/man1/sysbench.1.html
4. http://manpages.ubuntu.com/manpages/trusty/man8/powertop.8.html

TABLE 1: Resource usage for single request of resource information.

Device	Configuration	RAM (%)	Processor (%)	Battery (%)	Timespent (seconds)
Raspberry Pi	CPU: 1.2 GHZ quad-core ARM Cortex A53 Memory: 1 GB SDRAM Network: 10/100 MBPS Ethernet, Wireless LAN	0.4	1.8	0.00082	0.0009
Mobile Device	CPU: 1.8GHz octa-core Memory: 2GB LPDDR3, Network:Wireless LAN (802.11n)	0.6	2.3	0.01052	0.095
Lap top	CPU: i7-7600U CPU @ 2.80GHz (Intel(R) Core(TM)) Memory:3GB (LPDDR4) , Network: 811b/g/n and Ethernet	0.1	1.3	0.00055	0.04
Server	CPU: Quad core AMD Opteron 63xx class CPU 3.3 GHZ Memory: 12GB	0.2	0.3	n/a	0.008

TABLE 2: Resource usage for single request of resource information with network and IoT protocols [Aditya 2013]

	Characterisitcs	HTTP		MQTT	
		3G	WIFI	3G	WIFI
Received Messages	Messages per Hour	1708	3628	160278	263314
	Battery Percentage per Hour	18.43%	3.45%	16.13%	4.23%
	Battery Percentages per Messages	0.01709	0.00095	0.0001	0.00002
	Received Messages	240/1024	524/1024	1024/1024	1024/1024
Sent Messages	Messages per Hour	1926	5229	21685	23184
	Battery Percentage per Hour	18.80%	5.45%	17.81%	3.66%
	Battery Percentages per Messages	0.00975	0.00104	0.00082	0.00016

and Fog leaders which maximises the efficiency of resource information. To keep track of dynamic resource information, the proposed technique requests the Fog devices to update their resource information for the rejoin events of Fog devices. The remaining part of this section explains the each level agent algorithm in detail.

Notations:

- GETINFO OR GETPRESENTRESINFO: retrieves of FDId, RAM usage, CPU usage, bandwidth usage, battery percentage, resource updated time, number of services executed, service names and location.
- GETLASTMONRESOURCEINFO: retrieves the last updated resource info of the Fog device.
- GETPRESRESINFO: retrieves the live resource info of the Fog device.
- *devicelist or liveFDList or getLiveFogDevices*: maintains the resource information of active fog devices which registered under the Fog Leader.
- *deviceliveresinfo*: maintains the live Fog devices resource info who registered under the Fog Leader.
- GETREGISTEREDFDS: gets the total list of registered Fog devices under the Fog Leader.
- GETMODIFICATIONTIME: gets the last updated time of Fog devices resource info to the Fog Leader.
- *predictedLiveDeviceList*: maintains the predicted Fog devices list.
- *COI*: Change of Interval.
- *lc*: location. *ts*: time span. *Slist*:serving FDs list.
- *NFR*: Number of times it has failed to serve the request and
- *TSR*=Total service requests received

The Fog device updates the resource information to Fog leader in following conditions

1) During the registration time.
2) Immediate after joining the network.
3) After the device restarted.
4) After service completes.
5) When the battery is low.

The above conditions are used for register and unregister the Fog devices from Fog leaders.

Algorithm 1 SCB Resource Monitoring Algorithm for Fog Device

1: **procedure** RESOURCFDMONITORING(ReqType)
2: *presresinfo* $\leftarrow$ GETPRESRESINFO()
3: **if** $ReqType \equiv$ "$UpdateInfo$" **then**
4: UPDATERESINFOTOFL($presresinfo, false$)
5: **else**
6: **if** $isRegistered() \| isNetworkRejoined() \| isDevicereStarted() \| isServiceCompleted()$ & $isBattery() != low$ **then**
7: UPDATERESINFOTOFL($presresinfo, true$)
8: **else**
9: **if** $isNewJoin() \| isUnpredictedFD()$ **then**
10: $COI \leftarrow \frac{Numberofchanges}{hour}$
11: $timer(COI)$
12: UPDATERESINFOTOFL($presresinfo, false$)
13: *lastmodificationtime* $\leftarrow$ presenttime

When the battery is low the device sends the resource information along with the status has false indicating that it is not ready to serve any requests(unregister from Fog leader) as shown in Algorithm 1. Most of the Fog devices are not dedicated to serving the Fog services due to which there is a frequent change in resources of Fog device may happen. Hence, continuous resource monitoring is not suitable for the Fog environment and to find the liveliness of the devices; the above events are used to know whether the device is ready for the service execution.

Algorithm 2 SCB Resource Monitoring Algorithm for Fog Server

procedure PREDICTEDDEVICEMETAINFO
  FDinfo ← *open('log.txt', 'r')*
  for each $FD \in \mathcal{F}Dinfo$ **do**
    lc ← $FD[2]$, *ts* ← $FD[3]$
    Support[FD:lc:ts] ←*increment the count of FD in lc which is available at ts*
    Support[FD] ←*increment the count of FD*
  for each $FDID \in \mathcal{S}upport[FDid]$ **do**
    Confidence[FDID:lc:ts] ← $\frac{Support[FDid:lc:ts]}{Support[FDid]}$
  PredFDlist ← $sort(confidence[FDid:lc:ts])$
  for each $FDid \in \mathcal{P}redlFDlist$ **do**
    metainfo(FDid:lc:ts) ← (GETINFO(FDID))
    $Unpredictableratio \leftarrow \frac{NFR}{TSR} \times 100$
    if $NFR \leq 80$ **then**
      $Unpredictedlist.put(FDid)$

procedure SUPPORTANDCONFIDENCE(Requestinfo)
  failurefactor(ff) ← $\frac{Noofmisspredictionsreq}{totalnoofpredictions} \times 100$
  FDpredlist ← *getPredicteddevices(reqinfo)*
  predictedFDs ← *FDpredlist.length* + ff
  for each $FDid \in \mathcal{F}Dpredlist$ **do**
    FDcount ← 0
    if $FDcount \leq predictedFDs \ \ \& \ \ FDid \neq Slist$ **then**
      predictedlist ← $livedevicemetainfo$
      FDcount ← *FDcount+1*
  CREATETHREAD(SPT,SPT1)
  MAINTHREAD.UPDATEDEVICELIST(PREDICTEDLIST)
  SPT.UPDATEDEVICERESOURCEINFO(PREDICTEDLIST)
  SPT1.RESMONITORINGVALIDATION
  if $FDcount \leq predictedFDs$ **then**
    for each $FDid \in \mathcal{U}npredictedlist$ **do**
      SPT.UPDATEDEVICERESOURCEINFO(FDID)
    remainFDs ← liveFDList+FDpredlist+Unpredictedlist
    regFDList ← $getRegisteredFDs()$
    Unreglist ← regFDList-remainFDs
    for each $FDid \in \mathcal{U}nreglist$ **do**
      SPT1.UPDATEDEVICERESOURCEINFO(FDID)

If the device is newly registered or behaving unexpectedly(unpredicted behaviour) in the Fog environment, then we closely observe the behaviour of Fog devices by updating the resource information on change of interval time. The change of interval is defined as the number of changes in the resources over a period of time with which the newly registered Fog devices behaviour can be obtained. Fog user requests the service to the Fog Server. The Fog Server checks whether the live devices are available to process the request. If the live devices are unable to process then for remaining devices Fog server sends the resource information request to the predicted devices which are not participating in the service execution. This is ensured by comparing with the resource allocated list of the services. By using a speculative thread, Fog server sends the requests for resource information to predicts the list of devices and live devices. Main thread continues its process by submitting the service request to the Fog leader as shown in Algorithm 2. Whenever the Fog server has sent a speculative thread for device info to the Fog devices, it collects and sends its device information to the Fog leader. If predicted devices have failed to serve the service request of the Fog client.

Algorithm 2, is based on support and confidence in the Fog Server. The time series log contains the following information Service id, Start time, End time Device id, Location, Processing or Execution time, CPU usage, Storage usage, Network usage, battery usage and Input request. From this log, the device liveliness and resource information are provided based on support and confidence. This information is computed along with the failure factor. Which is the number of mispredicted devices for each request. The Fog server requests unpredicted and unregistered devices in that particular location to update their resource information to their respective Fog leaders. Unpredicted devices are calculated using the number of times it has failed to serve the request per total received service requests. If the devices respond to the Fog leader failed to serve the execution, the Fog leader requests the neighbour Fog leaders to get the required number of devices to serve the request. If it is still unable to process the request, it assigns the request to the Cloud. The resource monitoring validation as shown in Algorithm 3 will run dynamically whenever the service request has been received from the Fog client to maintain the live device information up-to-date. Resource monitoring validation is initiated from the Fog server.

Algorithm 3 SCB Resource Monitoring Algorithm for Fog Leader

procedure UPDATERESINFOTOFL(devresinfo,livestatus)
  if $livestatus$ **then**
    $deviceliveresourceinfo.put(devresinfo)$
  else
    $deviceliveresourceinfo.remove(devresinfo)$

procedure RESMONITORINGVALIDATION
  FDregisteredlist ← GETREGISTEREDFDS()
  for each $FDid \in \mathcal{F}Dregisteredlist$ **do**
    lastmodtime ← GETMODIFICATIONTIME(FDID)
    presenttime ← GETPRESENTTIME()
    period ← presenttime − lastmodtime
    if $period \geq invalidationtime$ **then**
      liveFD List ← REMOVEFROMFDLIST(FDID)

procedure UPDATEPREDICTEDLIST(DEVICELIST)
predictedLiveDeviceList.put(DEVICELIST)

5 INVESTIGATED RESOURCE MONITORING MODELS

In this section, traditional resource monitoring models like push-based, pull-based and hybrid algorithms and subsequently proposed resource monitoring models are discussed in detail.

5.1 Push-based

We adapted Pull-based model [45] previously used in Cloud computing for Fog computing. In the push-based model, the resource monitoring agent algorithm runs in every Fog device in the Fog environment. An agent is responsible for pushing the resource information from Fog devices to the Fog leader based on frequent intervals specified. If there is any change in the resource utilisation, it updates the latest resource information to Fog Leader. Otherwise, it will send only a message as *"I am alive"* with Fog device id as shown in Algorithm 4.

Algorithm 4 Push-based Resource Monitoring Algorithm at Fog device

```
1: procedure RESOURCEMONITORING(timeinterval)
2:     while true do
3:         timer(timeinterval)
4:         if timer.isFinished() then
5:             lastmoninfo ← GETLASTMONRESINFO()
6:             presentresinfo ← GETPRESENTRESINFO()
7:             if presentresinfo ≡ lastmoninfo then
8:                 HEARTBEATMSG("I AM ALIVE",FDID)
9:             else
10:                UPDATERESINFO(PRESENTRESINFO)
```

The Fog leader agent algorithm collects the information that has been received from the Fog devices, and it will update and maintain the live device list and resource information with *HEARTBEATMESSAGE procedure* and *UPDATERESOURCEINFO* as shown in Algorithm 5.

Algorithm 5 Push-based Resource Monitoring Algorithm at Fog Leader

```
1: procedure HEARTBEATMSG(msg,Fogdeviceid)
2:     devicelivelist.put(Fogdeviceid)
3:
4: procedure UPDATERESINFO(deviceinfo)
5:     deviceliveresinfo.put(deviceinfo)
```

5.2 Pull-based

The pull-based algorithm is also known as the dynamic algorithm because the resource information is obtained dynamically whenever the request has been received. We adapted Pull-based model [46] previously used in Cloud computing for Fog computing. In the pull-based algorithm, as shown in Algorithm 6, the resource information for Fog devices is requested whenever The Fog leader receives a service request from the cloud.

Algorithm 6 Pull-based Resource Monitoring Algorithm at Fog Leader

```
1: procedure RESOURCEMONITORING(ServiceRequest)
2:     if isServiceRequest() then
3:         Fogregisteredlist ← GETREGISTEREDFDS()
4:         if liveDeviceList.size() ≤ numberofreqdevices
   then
5:             for each FDid ∈ Fogregisteredlist do
6:                 deviceresinfo ← requestResInfo(Fogid)
7:
8: procedure UPDATERESINFO(deviceinfo)
9:     deviceliveresourceinfo.put(deviceinfo)
```

Algorithm 7 Pull-based Resource Monitoring Algorithm at Fog device

```
1: procedure REQUESTRESINFO(Fogid)
2:     presentresinfo ← getPresentResInfo()
3:     UPDATERESINFO(PRESENTRESINFO)
4:     lastmodificationtime ← presenttime
```

The *getRegisteredFogDevices* procedure retrieves the list of Fog devices that are registered to that particular leader, requests all Fog devices to send their device resource information as shown in algorithm 7, the devices send their present resource information to a Fog Leader. Fog Leader will update the resource information and maintains the list.

5.3 Hybrid-based

The combination of push and pull based model is known as a hybrid model. We adapted Hybrid model [47] previously used in Cloud computing for Fog computing. In Algorithm 8, the resource monitoring agent updates any changes of resources if they were last updated is more than time interval prior. If there is no change in the Fog device, the agent will not send any information about the Fog device, The push-based agent is deployed in Fog Leader and the pull-based model is deployed in Fog devices. If there is a change in the resources, The resource information will be updated, but if there are no updates for a specified time interval, it will pull the resources from the respective Fog devices.

Algorithm 8 Hybrid-based Resource Monitoring Algorithm Fog device

```
procedure RESOURCEMONITORING
    lastmoninfo ← GETLASTMONITOREDRESOURCEINFO()
    presentresinfo ← GETPRESENTRESOURCEINFO()
    if presentresinfo ≠ lastmoninfo then
        presenttime ← GETPRESENTTIME()
        period ← presenttime − lastmodificationtime
        if period ≥ timeinterval then
            UPDATERESOURCEINFOTOFL(PRESENTRESINFO)
            lastmodificationtime ← presenttime

procedure UPDATEDEVICERESOURCEINFO(Fogid)
    presentresourcinfo ← GETPRESENTRESOURCEINFO()
UPDATERESOURCEINFOTOFL(PRESENTRESOURCEINFO)
    lastmodificationtime ← presenttime
```

In Algorithm 9, if the Fog leader receives a request from the cloud and if the number of live devices is less than the required number of fog devices to serve the request, the Fog leader requests the unregistered list of devices to update the information. The *UPDATERESOURCEINFO* procedure will retrieve the present info and update the information to the Fog leader as shown in algorithm 8. In parallel, the *RESOURCEMONITORINGVALIDATION* procedure will

Algorithm 9 Hybrid-based Resource Monitoring Algorithm Fog Leader

procedure RESOURCEMONITORING(reqdevices)
  if $isServiceRequest()$ **then**
    liveDeviceList ← GETLIVEFOGDEVICES()
    registeredDeviceList ← GETREGISTEREDFDS()
    Unregisteredlist ← registeredDFDList-liveFDList
    if $liveDeviceList.size() \leq reqdevices$ **then**
      for each $FDid \in \mathcal{U}nregisteredlist$ **do**
        FDesourceinfo ← REQUESTRESINFO(FDID)

procedure RESOURCEMONITORINGVALIDATION
  FD*registeredlist* ← GETREGISTEREDFDS()
  for each $FDid \in \mathcal{F}Dregisteredlist$ **do**
    lastmodtime ← GETMODIFICATIONTIME(FDID)
    presenttime ← GETPRESENTTIME()
    period ← presenttime − lastmodtime
    if $period \geq timeinterval$ **then**
      liveFDList ← REMOVEFROMFDLIST(FDID)

procedure UPDATERESOURCEINFOTOFL(deviceinfo)
  deviceliveresinfo.put$(deviceinfo)$

check each device last updated time of resource information. If the Fog device new updated value is higher than the time interval specified, the Fog leader treats the device as dead, or there are no changes in their resources. The Fog leader updates the live device list and the unregistered list by removing the devices from the live list and added them to the unregistered list.

6 EXPERIMENTAL AND RESULT ANALYSIS

This section discusses the experimental environment and the overview of the use case with result analysis.

6.1 Experimental Environment

We implement the aforementioned resource monitoring techniques on a Fog environment using a proposed Fog emulator. For the experiments, we use Nectar cloud instance with a configuration of 12 cores processor each core with 3.2 GHz, 48GB RAM and 128GB storage and instance running on Linux 3.2 and develop Fog emulator using Java to evaluate the resource monitoring approaches with resource utilisation differences in percentage as a metric. We use this emulator to compare our approach, Pull and Hybrid with the Push model.

We test the resource monitoring methods by considering the use case of a traffic management system. The application considers traffic at a particular location or route. Fog user requests for the traffic service and specified locations, based on the user request, the Fog Server submits the request to respective Fog Leaders of the particular location. The Fog leader assigns Fog devices based on the number of traffic sensors in their respective locations. Fog devices collect the data from the sensors and validate the data. If the data is same as last updated data or invalid data, then data will be discarded. After capturing the data, the Fog leader aggregates the results from the other Fog devices and sends the results back to the user as shown in Figure 3. For example, Fog user requests a Fog server to provide traffic service in the location *Fog Colony-3*. The Fog Server receives the request from the Fog user and forwards this request to the Fog Leader in the location *Fog Colony-3* with predicted Fog devices list and Fog user details. Predicted number of devices are based on the number of sensors present in the requested location *Fog Colony-3*. The Fog leader receives the request information and activates the sensors and assigns to the predicted Fog devices as *FD-1*, *FD-2* and *FD-6*. These Fog devices will collect the data from the sensors and processed the data. The results processed by the Fog devices will be sent to the Fog leader, and which will send the results back to the user. If the same requests are received while processing the service, then it will not allocate the Fog devices separately it will wait until it process and sends the same results to all Fog users. The battery consumed and time taken to send or receive messages are calculated based on IoT protocols and the type of network from Table 2.

An open queue model followed the user requests for the services at every point of time in the emulator. This model maintains a fixed number of services (i.e. four services/requests). The submission of services will be random, and if the service is unable to serve by the fog devices in the same location, it requests the remaining devices from neighbour Fog leaders. If it still requires devices to serve the request, the remaining part of service will be computed in the cloud. We evaluated the resource usage, and the execution time of the resource monitoring service and the application service by varying the number of services and number of Fog devices in the environment. As the Fog devices are in either running or idle state, we considered the following two scenarios for resource monitoring:

1) The Fog devices are executing service requests, and
2) The Fog devices are not executing any service requests.

6.1.1 Performance metrics:

The following performance metrics are considered and present the resource usage of Fog devices in percentage resource utilisation difference with respect to the Push model, the number of Fog devices allocated in percentage difference of number of devices allocated in comparison with Push model and Fog leaders resource usage in the percentage of consumption.

- Device allocation: The number of devices that are used to serve the service requests successfully.
- Execution time: The running time of the service requests.
- Resource usage: The amount of CPU, RAM, Network and Battery consumed for the resource monitoring and service execution.
- Time spent: The amount of time spent on updating the resource information to Fog leader.

6.2 Fog Emulator

The Fog client sends a request to the Fog server to create the Fog environment in two ways, either in static and dynamic.

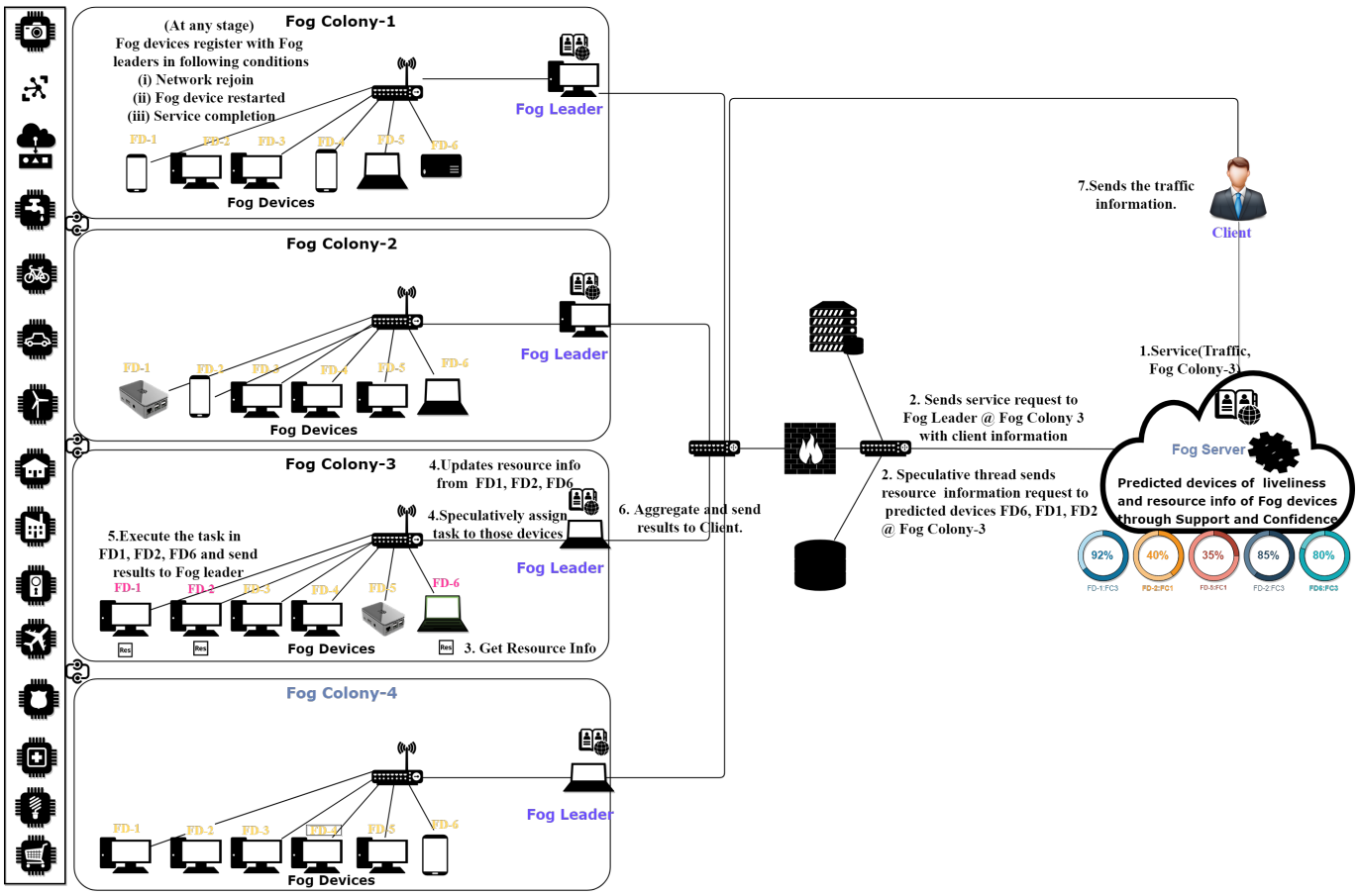


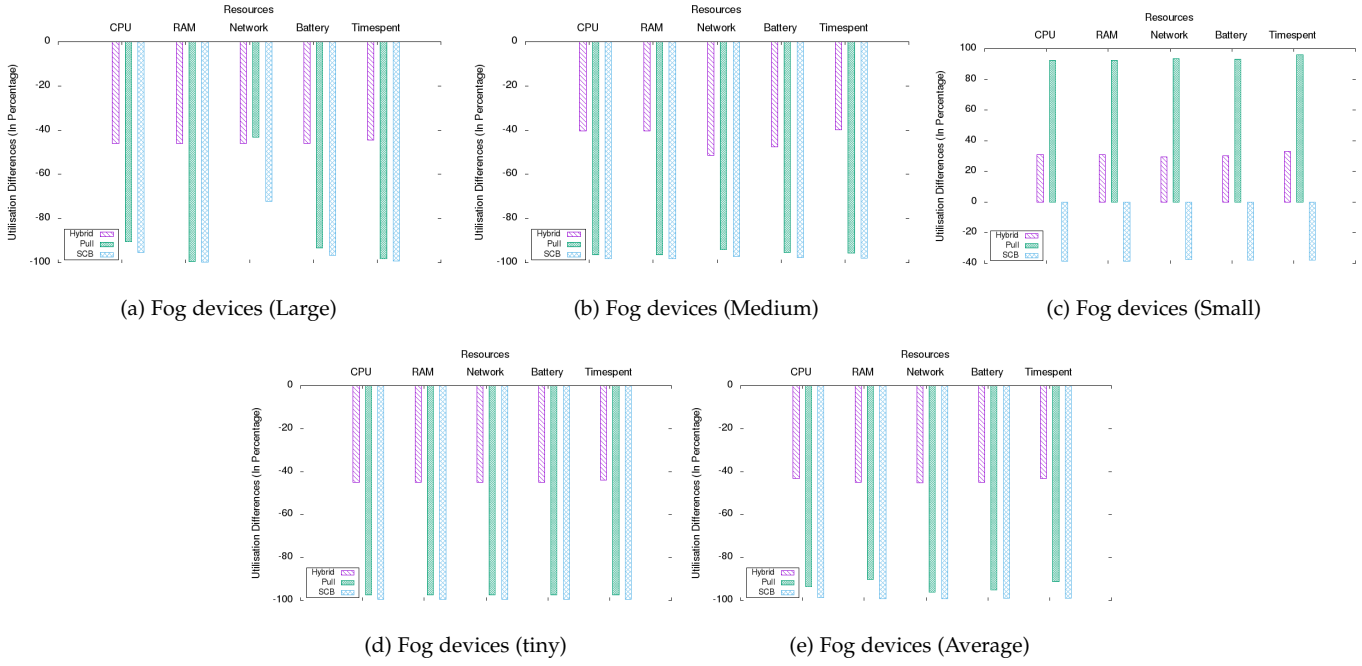
Fig. 3: Transportation use case in Fog environment

TABLE 3: Number of events processed for service in both container and non container systems.

	Containers			Bare-Metal		
Fog Device	Events per second	Average Latency (ms)	Maximum Latency (ms)	Events per second	Average Latency (ms)	Maximum Latency (ms)
Raspberry Pi	612.7075547	1.6	8.2	689.6	1.45	5.99
Nexus (Mobile Device)	-	-	-	800.6	1.45	5.99
Laptop	891.12	0.1600	9.01	9448.27	0.11	8.94
Server	3471106	0	1.37	384864.84	0	1.3

In the static mode, the client should specify each requirement of Fog environment such as number of locations, number of leaders and number of devices, device type, topology, number of sensors. Based on the input specifications, the Fog Server creates the environment. While in dynamic mode, users only need to specify minimum requirements such as a total number of locations, the maximum number of leaders in each location, the maximum number of devices and sensors. Based on those requirements, the Fog server creates the environment dynamically. Firstly, the Fog server creates the Fog Leaders with characteristics as ID, processors, processor speed, RAM, bandwidth, battery (optional), hard disk, location. The ID is the unique address to identify the Fog leader. It will be assigned by the Fog Server when it is registered with Fog server or when it is created. The *FogDynamicTopology* process creates a dynamic network between Fog devices, sensors and Fog leaders to complete the remaining process of creating the Fog environment. In order to develop dynamic behaviour in the environment, a *FogDynamicBehaviour* process is responsible for turning Fog devices and leaders on and off randomly to mimic the behaviour of real environments such as network failure, sudden shut-down of devices, and mobility of devices by changing the Fog device or leader location to another location with random intervals of time. Whenever Fog leaders fail in accepting a request, the Fog Server assigns another Fog leader from the available devices by analysing the device utilisation and service process utilisation from the log. The Fog server selects the device whose configuration and device live time is high and copies the Fog leader metadata from the server to elected Fog device. The Fog server is also responsible for accepting the service request from the users and submitting the request to the respective Fog leader by validating the details in Fog Server.

After the creation of a Fog leader by the Fog Server, the Fog leader receives the request from the Fog server on behalf of the Fog client. The Fog leader will start creating Fog Devices randomly with different types of Fog devices



(a) Fog devices (Large) (b) Fog devices (Medium) (c) Fog devices (Small)

(d) Fog devices (tiny) (e) Fog devices (Average)

Fig. 4: Percentage resource utilisation difference of Fog devices in resource monitoring with respect to Push model

such as Meshlium or Raspberry Pi and Mobile devices, Laptop, and server. We considered these devices based on the configuration and categorised as tiny, small, medium, large as specified in Fog Computing Surveys [8], [48]. Fog server creates network topology dynamically with the random distance between each Fog Devices in hierarchical fashion the root node will always be the Fog leader. This process is known as Fog device registration, Each Fog device will directly or indirectly connect to the Fog leaders, and all Fog leaders are connected to Fog servers. Finally, it creates sensors randomly without exceeding the Fog client requests in each location with the following characteristics: sensor id, sensor name; type of sensor; protocol; network; battery percentage; location, data (random values). The Fog leader updates the registered device information to Fog Server. Data will always flow from Fog Device to Fog Leader and Fog Leader to Fog Server or Fog user.

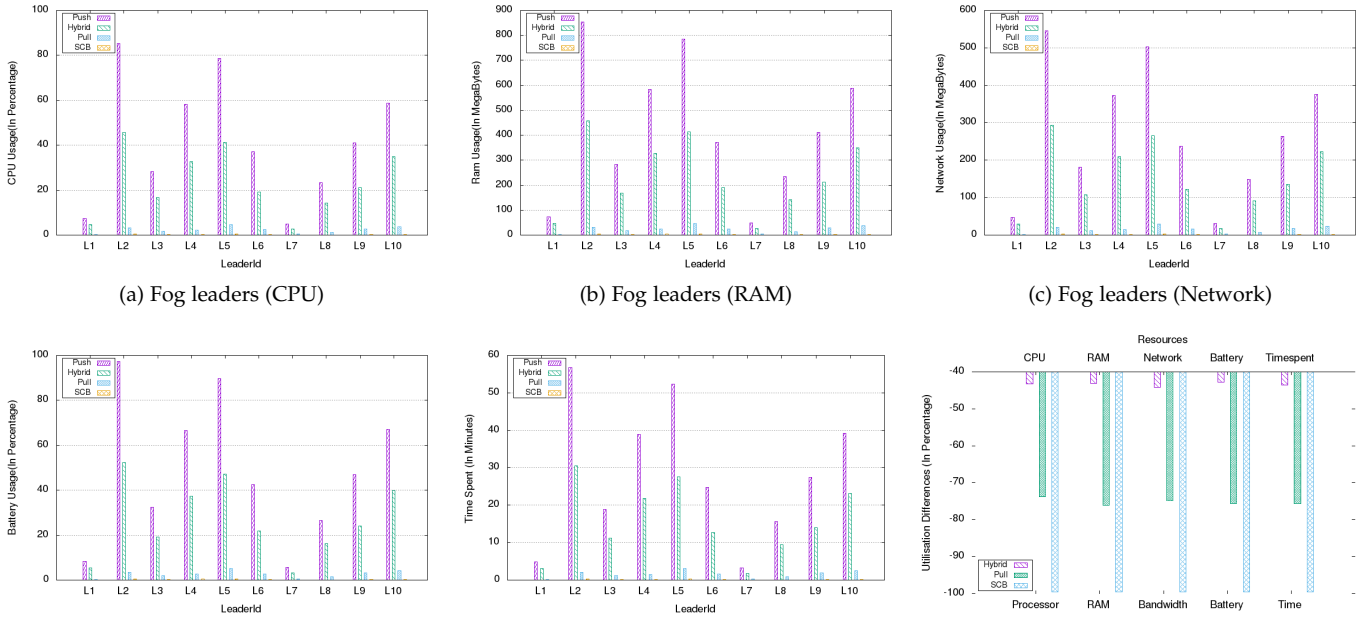
6.3 Results Analysis

The Fog environment in our experiment consists of 10 different locations with a maximum of 200 devices, 1000 sensors with at-least one Fog leader in each location and 200 service requests. The submission of applications followed an open queue model in such a way that, four services are running at any given instant of time.

6.3.1 Scenario 1: The Fog devices are executing service requests

The resource utilisation of different models were compared against the Push model during the execution of the services for the large Fog devices which showed SCB model used about 95% less CPU, about 99% less RAM, about 72% less Network, about 96% less Battery and about 99% less time than the Push model as shown in Figure 4a. However, the Pull and Hybrid model has more resource consumptions than the proposed SCB model. This is because of the least occurrence of update of resource information in proposed SCB model as compared to frequent and repetitive updates in other models. The resource consumption of medium and tiny Fog devices follows the same pattern for different models as shown in the Figure 4b and Figure 4d respectively. But for the small Fog devices, the Hybrid and Pull model have inferior performance in resource utilisation than Push based model because of the frequent resource changes in the Fog device and number of service requests during the experiment in the environment which resulted in more number of resource information requests. Thus the SCB model has the least resource consumption in Fog devices compare to the other models as shown in the Figure 4e.

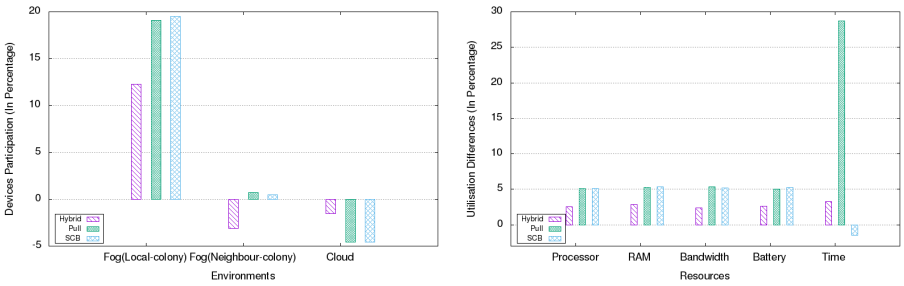
The CPU utilisation, RAM utilisation, network utilisation, battery Consumption and time spent of Fog leaders at different locations for the SCB model was significantly less when subjected to a high number of devices and service requests compared to other models (Figure 5a-Figure 5e). The pull-based model has the highest resource usage because of the frequent updates and heartbeat messages sent to the Fog leaders whereas, in Pull model, Fog leader sends the request to all unavailable nodes or unregistered nodes only when it does not have enough number of devices to serve the requests. This results in the less resource consumption compared to the push model. The hybrid model uses both push and pull model due to the dynamic behaviour or the frequent changes of Fog devices within a time is more, which contributes to the larger number of heart beat messages sent to the Fog leader. However, the proposed model only selects and gets the resource information from the predicted nodes and hence the number of request updates of resource information is very less. On average, the resource consumption of the proposed model in Fog leader is lower



(a) Fog leaders (CPU) (b) Fog leaders (RAM) (c) Fog leaders (Network)

(d) Fog leaders (Battery) (e) Fog leaders (Time) (f) Fog leaders (Average)

Fig. 5: Percentage resource utilisation of Fog leaders in resource monitoring



(a) Number of devices allocated (b) Fog device (Service)

Fig. 6: Percentage resource utilisation difference of Fog devices in service execution with respect to Push model

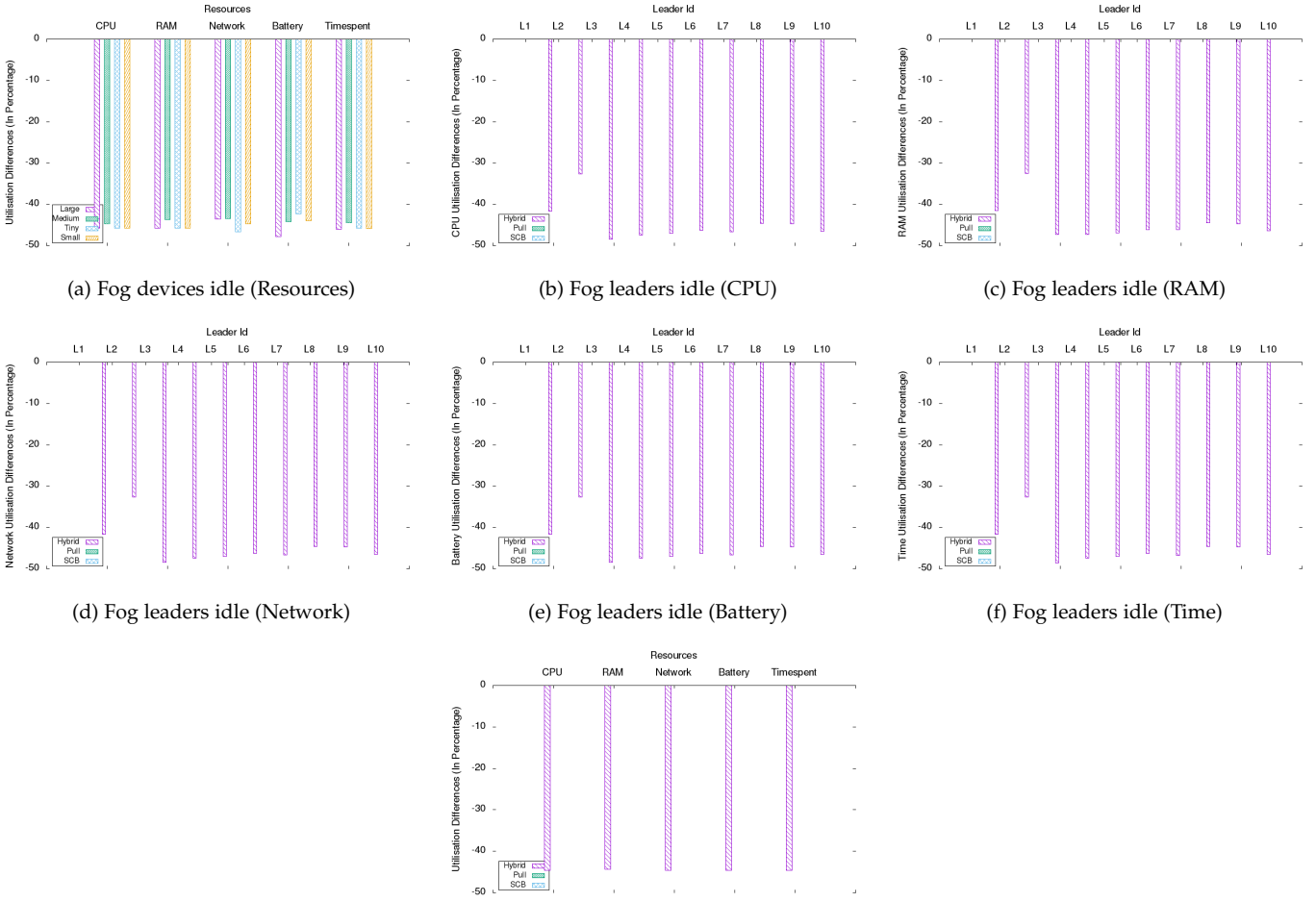
than that of Hybrid model, Push model and Pull model by about 52%, 98% and 24% respectively (Figure 5f. Thus, the proposed model is better than other models in terms of resource utilisation.

The number of devices allocated in Fog environment for executing the service requests compared with the push model is shown in Figure 6a. The number of nodes assigned for the services in Push model models has comparatively no difference with Pull and SCB. However, the pull-based model has assigned the highest number of local Fog devices to serve the request of the client because of the dynamic updates from the Fog device to the Leader where it consumes more resource usage of Fog device whenever there is a delay in receiving the service requests. The least number of nodes are assigned in the push model because of dynamic behaviour in the environment. Our proposed model also assigned an almost equal number of Fog devices in local, neighbour and cloud. The fewer difference of Fog devices are assigned because of unpredictable behaviour of Fog devices, and the newly joined Fog devices are participating in the environment, and there is no log information available to predict. Hence, no need to keep track of all devices continuously only those devices whose behaviour is unpredicted.

For the new devices, we are keeping track of the behaviour of the device by the hybrid model by adjusting the timer has the change of interval which optimises the resource overhead even for newly joined devices. Figure 6b shows the resource usage difference of service execution and the pull-based model has taken more time than the other models almost 38% more time than the Push model. Hence the SCB model is efficient when compared with the other models.

6.3.2 Scenario 2: The Fog devices are not executing service requests

In this scenario where no service requests were received at all in Fog environment, the Pull model and SCB model had no resource usage (Figure 7a) in Fog devices because, in both models, resource information is only updated in Fog leader after receiving a service request. But in Hybrid and Push model, Fog devices send the resource information to the Fog leader consuming some resources even if there are no



(a) Fog devices idle (Resources)

(b) Fog leaders idle (CPU)

(c) Fog leaders idle (RAM)

(d) Fog leaders idle (Network)

(e) Fog leaders idle (Battery)

(f) Fog leaders idle (Time)

(g) Fog leaders (Average)

Fig. 7: Percentage resource utilisation difference of Fog devices and leaders in resource monitoring with respect to Push model when devices are idle

service requests in the Fog environment. The Hybrid model has about 46% lesser resource utilisation than the Push model due to the fewer updates of resource information to the Fog leader than the Push model. The hybrid model only updates if there is any change in the resource with the last updated one.

Figure 7b-7f shows the CPU, RAM, network, battery and time utilisation differences in percentages respectively when no service requests are executing. Resource usage has been varying from one leader to another is due to the services, and the number of devices in each location associated with the leaders are different. In Pull and SCB model the Fog leader initiates the resource information requests to the Fog devices only when it receives the services. But whereas in the Hybrid model the Fog devices update the resource information to Fog leaders only when there is any change in Fog device resources. Moreover, in the Push model the Fog devices update the resource information to Fog leaders in frequent intervals due to which it consumes more Fog leader resources than the Hybrid model. The average resource usage and time spent of Fog leaders are presented in Figure 7g shows that the resource usage in the Push model is more than 43% of the Hybrid model.

To conclude, the SCB model has less resource overhead in both Fog devices and Fog leaders for resource monitoring service even when the number of Fog devices and services are increased compared to the traditional approaches. The limitation of this model is that it consumes some resources from the cloud for analysing the log to calculate support and confidence for each device. However, the cloud has infinite virtual and physical resources. Hence, the proposed resource monitoring service doesn't have any burden on resource monitoring.

6.4 Discussion

We compared the resource monitoring models and revealed that the push model provides up-to-date resource information, and better efficiency of services will be provided. This accuracy of information helps in preventing the failure of a request while scheduling. The limitation of this model is that it continuously keeps track of resource information and updates to Fog leader at a fixed time interval. Hence, both Fog devices and Fog leaders are wasting many resources on monitoring. The pull model dynamically gets the resource information whenever the Fog Leader receives the request due to which, the time it takes to collect the information

and then allocate resource for the service is more. Hence, it takes a longer time to serve the request. So this model is not suitable for the time-sensitive applications in Fog computing. Most of the devices in the Fog environment are not dedicatedly allocated to serve the requests of the users and also not static. Due to which the device resource information keeps changing because of that the resource consumption for monitoring is also high in this model. The proposed model does not continuously keep track of resources, and It will only send a request for resource information to selected nodes based on the Confidence of the device. So the proposed resource monitoring technique has fewer resource consumption than the conventional models.

Monitoring tools are not required to continuously monitor the resources in Fog environment because the Fog environment does not have a dedicated node to process the data. Hence, using the agent continuously tracking the information is not a good idea. The proposed model will work based on the support and confidence of the device to know whether the device is ready now to take the task and also adjusting the failure factor dynamically to handle even some of the incorrect predictions.

7 Conclusion and Future Directions

The massive rise in the use of IoT devices in different applications of Fog computing has made the resource monitoring task hugely complex. In this paper, we investigated various resource monitoring techniques of distributed systems and concluded that existing resource monitoring techniques of other fields are not suitable for Fog environments due to their characteristics. Resources should not be continuously monitored because of limited resources. Hence, we proposed an SCB resource monitoring model, evaluated and compared the results using a Fog emulator. Our results show that our proposed technique has less resource consumption in both Fog devices and Fog leaders without significant difference in performance compared to a Push-based model.

In terms of future work, researchers can build a scalable Fog computing platform with advanced services on top of the proposed resource monitoring service to improve the overall performance of the system.

Acknowledgments

The authors would like to sincerely thank the Nectar Research Cloud for providing instances and Smart Service Systems Lab (UTAS) members for their insightful comments and discussion on improving the overall quality of the paper. The first author would like to thank the University of Tasmania (UTAS) for providing Tasmania Graduate Research Scholarship (TGRS) for supporting his studies.

References

[1] L. Li, H. Xiaoguang, C. Ke, and H. Ketai, "The applications of wifi-based wireless sensor network in internet of things and smart grid," in *Industrial Electronics and Applications (ICIEA), 2011 6th IEEE Conference on*. IEEE, 2011, pp. 789–793.

[2] S. Tammishetty, T. Ragunathan, S. K. Battula, B. V. Rani, P. RaviBabu, R. Nagireddy, V. Jorika, and V. M. Reddy, "Iot-based traffic signal control technique for helping emergency vehicles," in *Proceedings of the First International Conference on Computational Intelligence and Informatics*. Springer, 2017, pp. 433–440.

[3] M. Lacey, H. Lisachuk, A. Giannopoulos, and A. Ogura, "Shipping smarter: Iot opportunities in transport and logistics," *An article in Deloittes series examining the nature and impact of the Internet of Things*, 2015.

[4] F. TongKe, "Smart agriculture based on cloud computing and iot," *Journal of Convergence Information Technology*, vol. 8, no. 2, 2013.

[5] A. Nordrum, "Popular internet of things forecast of 50 billion devices by 2020 is outdated," *IEEE Spectrum*, vol. 18, 2016.

[6] D. R. de Vasconcelos, R. M. de Castro Andrade, and J. N. de Souza, "Smart shadow–an autonomous availability computation resource allocation platform for internet of things in the fog computing environment," in *Distributed Computing in Sensor Systems (DCOSS), 2015 International Conference on*. IEEE, 2015, pp. 216–217.

[7] F. C. Delicato, P. F. Pires, and T. Batista, "The resource management challenge in iot," in *Resource Management for Internet of Things*. Springer, 2017, pp. 7–18.

[8] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos, "Fog computing for sustainable smart cities: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, p. 32, 2017.

[9] O. Skarlat, K. Bachmann, and S. Schulte, "Fogframe: Iot service deployment and execution in the fog," *KuVS-Fachgespräch Fog Computing 2018*, p. 5, 2018.

[10] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, 2017.

[11] A. CloudWatch, "monitoring for aws cloud resources," *Retrieved May*, vol. 7, p. 2015, 2013.

[12] Cloudmonix, "CLOUD AND ON-PREMISE MONITORING AND AUTOMATION," http://www.cloudmonix.com/aw/, 2015, [Online; accessed 19-June-2018].

[13] G. Karjoth, "Access control with ibm tivoli access manager," *ACM Transactions on Information and System Security (TISSEC)*, vol. 6, no. 2, pp. 232–257, 2003.

[14] R. K. Naha, S. Garg, D. Georgekopolous, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, "Fog computing: Survey of trends, architectures, requirements, and research directions," *Distributed, Parallel, and Cluster Computing*, 2018. [Online]. Available: https://arxiv.org/abs/1807.00976

[15] M. Abderrahim, M. Ouzzif, K. Guillouard, J. Francois, and A. Lèbre, "A holistic monitoring service for fog/edge infrastructures: a foresight study," in *The IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud 2017)*, 2017.

[16] P.-H. Tsai, H.-J. Hong, A.-C. Cheng, and C.-H. Hsu, "Distributed analytics in fog computing platforms using tensorflow and kubernetes," in *Network Operations and Management Symposium (APNOMS), 2017 19th Asia-Pacific*. IEEE, 2017, pp. 145–150.

[17] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[18] V. B. Souza, X. Masip-Bruin, E. Marin-Tordera, W. Ramírez, and S. Sanchez, "Towards distributed service allocation in fog-to-cloud (f2c) scenarios," in *Global Communications Conference (GLOBECOM), 2016 IEEE*. IEEE, 2016, pp. 1–6.

[19] M. Aazam and E.-N. Huh, "Fog computing micro datacenter based dynamic resource estimation and pricing model for iot," in *Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on*. IEEE, 2015, pp. 687–694.

[20] S. Zanikolas and R. Sakellariou, "A taxonomy of grid monitoring systems," *Future Generation Computer Systems*, vol. 21, no. 1, pp. 163–188, 2005.

[21] R. Sundaresan, T. Kurc, M. Lauria, S. Parthasarathy, and J. Saltz, "A slacker coherence protocol for pull-based monitoring of on-line data sources," in *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*. IEEE, 2003, pp. 250–257.

[22] W.-C. Chung and R.-S. Chang, "A new mechanism for resource monitoring in grid computing," *Future Generation Computer Systems*, vol. 25, no. 1, pp. 1–7, 2009.

[23] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *High*

Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on. IEEE, 2001, pp. 181–194.
[24] B. Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, M. Swany *et al.*, "White paper: A grid monitoring service architecture (draft)," in *Global Grid Forum*, 2001.
[25] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*. Ieee, 2008, pp. 1–10.
[26] I. Legrand, C. Cirstoiu, C. Grigoras, R. Voicu, M. Toarta, C. Dobre, and H. Newman, "Monalisa: An agent based, dynamic service system to monitor, control and optimize grid based applications," 2005.
[27] E. Imamagic and D. Dobrenic, "Grid infrastructure monitoring system based on nagios," in *Proceedings of the 2007 workshop on Grid monitoring*. ACM, 2007, pp. 23–28.
[28] X. Xu, Y. Chen, and J. M. A. Calero, "Distributed decentralized collaborative monitoring architecture for cloud infrastructures," *Cluster Computing*, vol. 20, no. 3, pp. 2451–2463, 2017.
[29] A. Anand, M. Dhingra, J. Lakshmi, and S. Nandy, "Resource usage monitoring for kvm based virtual machines," in *Advanced Computing and Communications (ADCOM), 2012 18th Annual International Conference on*. IEEE, 2012, pp. 66–70.
[30] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, "The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds," *Future Generation Computer Systems*, vol. 28, no. 6, pp. 861–870, 2012.
[31] C. Nimsoft, "Unified monitoring," *Nimsoft*, vol. 1, no. 1, p. 10, 2011.
[32] J. Povedano-Molina, J. M. Lopez-Vega, J. M. Lopez-Soler, A. Corradi, and L. Foschini, "Dargos: A highly adaptable and scalable monitoring architecture for multi-tenant clouds," *Future Generation Computer Systems*, vol. 29, no. 8, pp. 2041–2056, 2013.
[33] S. A. De Chaves, R. B. Uriarte, and C. B. Westphall, "Toward an architecture for monitoring private clouds," *IEEE Communications Magazine*, vol. 49, no. 12, pp. 130–137, 2011.
[34] N. Sabharwal, *Apache cloudstack cloud computing*. Packt Publishing Ltd, 2013.
[35] D. Milojičić, I. M. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," *IEEE Internet Computing*, vol. 15, no. 2, pp. 11–14, 2011.
[36] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "Harmony: Dynamic heterogeneity-aware resource provisioning in the cloud," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. IEEE, 2013, pp. 510–519.
[37] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D. Patterson, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," in *Proc. of CCA*, vol. 8, 2008.
[38] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. D. Berger, "Eon: a language and runtime system for perpetual systems," in *Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM, 2007, pp. 161–174.
[39] K. Lorincz, B.-r. Chen, J. Waterman, G. Werner-Allen, and M. Welsh, "Resource aware programming in the pixie os," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 2008, pp. 211–224.
[40] S. Kang, Y. Lee, C. Min, Y. Ju, T. Park, J. Lee, Y. Rhee, and J. Song, "Orchestrator: An active resource orchestration framework for mobile context monitoring in sensor-rich mobile environments," in *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*. IEEE, 2010, pp. 135–144.
[41] B. Li and K. H. Wang, "Nonstop: Continuous multimedia streaming in wireless ad hoc networks with node mobility," *IEEE journal on Selected Areas in Communications*, vol. 21, no. 10, pp. 1627–1641, 2003.
[42] Y.-C. Hu and D. B. Johnson, "Exploiting congestion information in network and higher layer protocols in multihop wireless ad hoc networks," in *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on*. IEEE, 2004, pp. 301–310.
[43] O. C. A. W. Group *et al.*, "Openfog reference architecture for fog computing," *OPFRA001*, vol. 20817, p. 162, 2017.
[44] A. Dutta, "why HTTP is not enough for Internet of Things," https://www.ibm.com/developerworks/community/blogs/mobileblog/entry/, 2013, [Online; accessed 19-June-2018].
[45] K. An, S. Pradhan, F. Caglar, and A. Gokhale, "A publish/subscribe middleware for dependable and real-time resource monitoring in the cloud," in *Proceedings of the Workshop on Secure and Dependable Middleware for Cloud Monitoring and Management*. ACM, 2012, p. 3.
[46] A. Brinkmann, C. Fiehe, A. Litvina, I. Lück, L. Nagel, K. Narayanan, F. Ostermair, and W. Thronicke, "Scalable monitoring system for clouds," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 2013, pp. 351–356.
[47] M. Al-Ayyoub, Y. Jararweh, M. Daraghmeh, and Q. Althebyan, "Multi-agent based dynamic resource provisioning and monitoring for cloud computing systems infrastructure," *Cluster Computing*, vol. 18, no. 2, pp. 919–932, 2015.
[48] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

Battula Sudheer Kumar received his Master of Technology degree in software engineering from Jawaharlal Nehru Technological University (JNTU), India in 2012. He is currently pursuing his Ph.D. studies on resource management in Fog computing environment with the University of Tasmania. His research interests includes Big data, distributed file systems, Cloud computing, Internet of Things (IoT), and Fog computing.

SAURABH GARG is currently a Lecturer with the University of Tasmania, Australia. He is one of the few Ph.D. students who completed in less than three years from the University of Melbourne. He has authored over 40 papers in highly cited journals and conferences. His research interests include resource management, scheduling, utility, grid computing, Cloud computing, green computing, and ad hoc networks.

JAMES MONTGOMERY (M11) received the B.Inf.Tech. (Hons.) degree from Bond University, Gold Coast, Australia, in 2000, and the Ph.D. degree in computer science from Bond University, in 2005. He is currently a Lecturer with ICT, University of Tasmania, Australia. He has previously held post-doctoral positions at the Swinburne University of Technology and the Australian National University. His research interests span evolutionary computation, machine learning, and web services.

Byeong Kang received the Ph.D. degree from the University of New South Wales, Sydney, in 1996. He was a Researcher Fellow with the Hitachi Advanced Research Laboratory, Japan, in 1995. From 1998 to 2000, he was an Advisor on Research and Development (Industry), South Korea. From 2000 to 2007, he was a Senior Lecturer with the University of Tasmania. He is currently an Associate Professor of computing and information systems with the University of Tasmania, Australia. His research interest includes artificial intelligence, expert system, knowledge acquisition, and Internet applications.